Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

14 October 2024

Algorithms & Data Structures — Exercise sheet 4 — HS 24

The solutions for this sheet are submitted on Moodle until 20 October 2024, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

The solutions are intended to help you understand how to solve the exercises and are thus more detailed than what would be expected at the exam. All parts that contain explanation that you would not need to include in an exam are in grey.

Master theorem. The following theorem is very useful for running-time analysis of divide-and-conquer algorithms.

Theorem 1 (master theorem). *Let $a, C > 0$ and $b \geq 0$ be constants and $T : \mathbb{N} \to \mathbb{R}^+$ a function such that for all even $n \in \mathbb{N}$,*

$$T(n) \leq aT(n/2) + Cn^b. \tag{1}$$

Then for all $n = 2^k$, $k \in \mathbb{N}$, the following statements hold

(i) If $b > \log_2 a$, $T(n) \leq O(n^b)$.

(ii) If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.[1]

(iii) If $b < \log_2 a$, $T(n) \leq O(n^{\log_2 a})$.

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If we instead have

$$T(n) \geq aT(n/2) + C'n^b, \tag{2}$$

then we can conclude that $T(n) \geq \Omega(n^b), T(n) \geq \Omega(n^{\log_2 a} \cdot \log n)$, and $T(n) \geq \Omega(n^{\log_2 a})$ in cases $(i), (ii)$, and (iii), respectively. Furthermore if (1) and (2) both hold (with possibly different constants $C \neq C'$), then similarly $T(n) = \Theta(n^b), T(n) = \Theta(n^{\log_2 a} \cdot \log n)$, and $T(n) = \Theta(n^{\log_2 a})$ in cases $(i), (ii)$, and (iii), respectively.

This generalizes some results that you have already seen in this course. For example, the (worst-case) running time of Karatsuba's algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so we have $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \leq O(n^{\log_2 3})$. Another example is binary search: its running time satisfies $T(n) \leq T(n/2) + 100$, so $a = 1$ and $b = 0 = \log_2 1$, hence $T(n) \leq O(\log n)$.

Exercise 4.1 *Applying the master theorem.*

[1] For this asymptotic bound we assume $n \geq 2$ so that $n^{\log_2 a} \cdot \log n > 0$.

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \mathbb{N}_0$). In the following, you are given a function $T : \mathbb{N} \to \mathbb{R}^+$ defined recursively and you are asked to find its asymptotic behavior by applying the master theorem.

(a) Let $T(1) = 1$, $T(n) = 4T(n/2) + 100n$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n^2).$$

Solution:

We can apply the master theorem with $a = 4$, $b = 1$ and $C = 100$. In this case, $b < \log_2 a$, and therefore we have $T(n) \leq O(n^{\log_2 a})$, i.e., $T(n) \leq O(n^2)$ since $\log_2 a = 2$.

An alternative way of seeing that $T(n) \leq O(n^2)$ is to repeatedly apply the recursive definition of T until we reach $n = 1$, i.e.,

$$\begin{aligned} T(n) &= 4T(n/2) + 100n \\ &= 16T(n/4) + 400n/2 + 100n \\ &= 64T(n/8) + 1600n/4 + 400n/2 + 100n \\ &= 4^{\log_2(n)} + \sum_{i=1}^{\log_2(n)} 100 \cdot 4^{i-1} \frac{n}{2^{i-1}} \\ &= 2^{2\log_2(n)} + 100n \cdot (2^{\log_2(n)} - 1) \end{aligned}$$

since we must expand T for $\log_2(n)$ times until we hit $n = 1$. Now the first term is n^2 and the second term is $100n^2 - 100n$.

(b) Let $T(1) = 5$, $T(n) = T(n/2) + \frac{3}{2}n$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n).$$

Solution:

We can apply the master theorem with $a = 1$, $b = 1$ and $C = \frac{3}{2}$. In this case, $b > \log_2 a$, and therefore we have $T(n) \leq O(n^b)$, i.e., $T(n) \leq O(n)$.

Now when expanding the sum, we get

$$\begin{aligned} T(n) &= T(n/2) + \frac{3}{2}n \\ &= T(n/4) + \frac{3}{4}n + \frac{3}{2}n \\ &= T(n/8) + \frac{3}{8}n + \frac{3}{4}n + \frac{3}{2}n \\ &= 5 + 3n \sum_{i=1}^{\log_2(n)} 2^{-i}. \end{aligned}$$

This term is in $O(n)$ since the sum is a geometric sum and thus at most a constant.

(c) Let $T(1) = 4$, $T(n) = 4T(n/2) + \frac{7}{2}n^2$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n^2 \log n).$$

Solution:

We can apply the master theorem with $a = 4$, $b = 2$ and $C = \frac{7}{2}$. In this case, $b = \log_2 a$, and therefore we have $T(n) \leq O(n^{\log_2 a} \cdot \log n)$, i.e., $T(n) \leq O(n^2 \log n)$.

An alternative way of seeing that $T(n) \leq O(n^2)$ is to repeatedly apply the recursive definition of T until we reach $n = 1$, i.e.,

$$\begin{aligned}
T(n) &= 4T(n/2) + \frac{7}{2}n^2 \\
&= 16T(n/4) + \frac{4 \cdot 7}{2}(n/2)^2 + \frac{7}{2}n^2 \\
&= 64T(n/4) + \frac{16 \cdot 7}{2}(n/4)^2 + \frac{4 \cdot 7}{2}(n/2)^2 + \frac{7}{2}n^2 \\
&= 256T(n/4) + \frac{64 \cdot 7}{2}(n/8)^2 + \frac{16 \cdot 7}{2}(n/4)^2 + \frac{4 \cdot 7}{2}(n/2)^2 + \frac{7}{2}n^2 \\
&= 4^{\log_2(n)} + \frac{7}{2}\sum_{i=1}^{\log_2(n)} 4^{i-1}\left(\frac{n}{2^{i-1}}\right)^2 \\
&= 2^{2\log_2(n)} + \frac{7}{2}\sum_{i=1}^{\log_2(n)} n^2.
\end{aligned}$$

The first term is in $O(n^2)$ and the second one in $O(n^2 \log(n))$.

Exercise 4.2 *Asymptotic notations.*

(a) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☐
$\log(n!) \geq \Omega(n^2)$	☐	☐
$n^k \geq \Omega(k^n)$, if $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

Solution:

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☒
$\log n! \geq \Omega(n^2)$	☐	☒
$n^k \geq \Omega(k^n)$, if $1 < k \leq O(1)$	☐	☒
$\log_3 n^4 = \Theta(\log_7 n^8)$	☒	☐

For (1) we can just look at the limit of $\sqrt{n}/\log(n)$ as $n \to \infty$. For (2) note that $\log(n!) \leq \log(n^n) = n\log(n)$. For (3) we note that e.g. n^2 is asymptotically smaller than 2^n. For (4), note that $\log_3(n^4) = 4\log_3(n) = \frac{4}{\ln(3)}\ln(n)$ and that $\log_7(n^8) = 8\log_7(n) = \frac{8}{\ln(7)}\ln(n)$ so the two terms only differ by a constant.

(b) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☐

Note that the last claim is challenge. It was one of the hardest tasks of the exam. If you want a 6 grade, you should be able to solve such exercises.

Solution:

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☒	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☒
$3n^4 + n^2 + n \geq \Omega(n^2)$	☒	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☒

For (1), consider the limit $\frac{f(n)}{g(n)}$. For (2), note that $\log_7(n^8) = \Theta(\log(n))$ but $\log_3(n^{\sqrt{n}}) = \Theta(\sqrt{n}\log(n))$. For (3), consider again the limit $\frac{f(n)}{g(n)}$.

The last claim can be verified as follows. Note that for all $n \geq 1$,

$$n! \geq 1 \cdot 2 \cdots n \geq \lceil n/10 \rceil \cdots n \geq \lceil n/10 \rceil^{0.9n} \geq (n/10)^{0.9n}\,.$$

Let's show that $(n/10)^{0.9n}$ grows asymptotically faster than $n^{n/2}$.

$$\lim_{n\to\infty} \frac{n^{n/2}}{(n/10)^{0.9n}} = \lim_{n\to\infty} 10^{0.9n} \cdot n^{-0.4n} = \lim_{n\to\infty} (10^{9/4}/n)^{0.4n} = 0\,.$$

Hence it is not true that $(n/10)^{0.9n} \leq O(n^{n/2})$ and so it is not true that $n! \leq O(n^{n/2})$.

Sorting and Searching.

Exercise 4.3 *Formal proof of correctness for Insertion Sort* **(1 point)**.

Algorithm 1 Insertion Sort (input: array $A[1 \ldots n]$).

```
for $j = 1, \ldots, n$ do
    if $j > 1$ then
        for $i = j - 1, \ldots, 1$ do
            if $A[i+1] < A[i]$ then
                Swap $A[i+1]$ and $A[i]$
```

Prove correctness of this algorithm by mathematical induction.

Hint: *Use the invariant $I(j)$: "After j iterations the first j elements are sorted."*

Solution:

We prove the invariant in the hint by mathematical induction on j.

- **Base Case.**
  We prove the statement for $j = 1$. We have that the first element is sorted with respect to the sublist of only the first element. Therefore $I(1)$ holds. Note that nothing happens with the first iteration since we do not have $j > 1$.

- **Induction Hypothesis.**
  We assume that the invariant is true for $j = k$ for some $k \in \mathbb{N}$, $k < n$, i.e. after k iterations the first k are sorted.

- **Inductive Step.**
  We must show that the invariant also holds for $j = k+1$. By the induction hypothesis the first k elements are sorted, i.e. at positions $A[1, \ldots, k]$. We now consider step $k+1$. If $A[k+1] \geq A[k]$ then our second for loop does nothing in the first step, and by the inductive hypothesis the list $A[1, \ldots, k]$ is sorted thus the list $A[1, \ldots, k+1]$ is sorted.

  Otherwise $A[k+1] < A[k]$. Let $c = A[k+1]$ be the value of the list at the time and let $0 \leq \ell \leq k$ be the largest index of list A such that $A[\ell] \leq c$ and $\ell = 0$ if no such element exists. In this case, we can see that the second for loop will keep swapping adjacent elements $A[i+1]$ and $A[i]$ for $i = k, \ldots, \ell + 1$. This is because our sublist is sorted so we always satisfy $c = A[i+1] < A[i]$ when $i \geq \ell + 1$ at iteration i. The rest of the inner for loop does nothing since $c = A[\ell+1] \geq A[i]$ for $i \leq \ell$.

  Now since the sublist $A[1, \ldots, \ell]$ is sorted since its unchanged, $A[\ell+2, \ldots, k+1]$ is equal to $A[\ell+1, \ldots, k]$ before the inner for loop and thus sorted, as well as $A[\ell] \leq A[\ell+1] = c \leq A[\ell+2]$ by definition of ℓ, we can see that $A[1, \ldots, k+1]$ is sorted so $I(k+1)$ holds.

By the principle of mathematical induction, $I(j)$ is true for all $j \in \mathbb{N}, j \leq n$. In particular, $I(n)$ holds, which means that after the first n iterations the first n elements are sorted. This shows that after n steps the array is sorted, which shows correctness of the Insertion Sort algorithm.

Guidelines for correction:

This exercise consists of one large induction part. Award 1/2 point if the base case and inductive hypothesis are set up properly and a partial proof of the inductive step is attempted. Award 1 point if the inductive step is correct.

Exercise 4.4 *Searching in a weirdly-sorted array* **(1 point)**.

Let $n \geq 2$ and suppose we are given an array $A[1 \dots n]$ containing n unique integers, that satisfies the following property: there is an integer $1 \leq k \leq n-1$ such that the subarrays $A[1 \dots k]$ and $A[k+1 \dots n]$ are sorted (in ascending order), and $A[n] < A[1]$. We call such an array *weirdly-sorted*, and we call k the *pivot*.

(a) Given a weirdly-sorted array $A[1 \dots n]$ containing n unique integers, provide an algorithm in pseudocode that finds the pivot $1 \leq k \leq n-1$ such that the subarrays $A[1 \dots k]$ and $A[k+1 \dots n]$ are sorted (in ascending order). The runtime of your algorithm should be at most $O(\log n)$.

Hint: *Be careful of edge-cases.*

Hint: *For an index $1 \leq m \leq n$, think of a simple condition involving $A[1], A[n]$ you could check to see if m is to the left or to the right of the pivot.*

Solution:

Let k be the pivot. Note that, for any $m > k$ (i.e., to the right of the pivot), we must have $A[m] \leq A[n] < A[1]$, while for any $m < k$ (to the left of the pivot), we must have $A[m] < A[k]$.

Therefore, the pivot can be found using the following recursive algorithm:

Algorithm 2 Find the pivot

```
function FINDPIVOT($A$, $i$, $j$)
    if $j = i$ then
        return $i$
    $m \leftarrow \lceil (i+j)/2 \rceil$                      ▷ Mid-point between $i$ and $j$
    if $A[m] < A[1]$ then                        ▷ $m$ is strictly larger than the pivot.
        return FINDPIVOT$(A, i, m-1)$            ▷ keep searching in the left half
    else                                         ▷ $m$ is smaller than or equal to the pivot
        return FINDPIVOT$(A, m, j)$              ▷ keep searching in the right half
Input: Weirdly-sorted array $A$ of length $n$.
Output: FINDPIVOT$(A, 1, n)$
```

To show that this algorithm really works in $O(\log n)$, you can use the Master theorem.

(b) Given a weirdly-sorted array $A[1 \dots n]$ containing n unique integers, and an integer $\ell \in \mathbb{N}$, provide an algorithm in pseudocode that determines whether A contains ℓ as an entry. The runtime of your algorithm should be at most $O(\log n)$. You may use the algorithm of part (a) as a subroutine even if you did not solve that part. You may also use algorithms from the lecture as subroutines.

Solution:

Consider the binary search algorithm BS for integer arrays sorted in ascending order. That is, given such an array B of length n, and an integer $p \in \mathbb{N}$, the algorithm $\mathrm{BS}(B, p)$ returns true if p is an entry of B and false otherwise. It has running time $O(\log n)$. The idea for searching in a weirdly-sorted array is to first find the pivot using the algorithm of part (a), and then apply binary search to both (sorted) halves of the array:

Algorithm 3 Search in a weirdly-sorted array

```
**Input**: Weirdly-sorted array $A$ of length $n$ with unique elements, integer $\ell$
$k \leftarrow \textsc{findPivot}(A, 1, n)$
$t_1 \leftarrow \mathrm{BS}(A[1 \dots k], \ell)$            ▷ search in array $A[1 \dots k]$, sorted in ascending order
$t_2 \leftarrow \mathrm{BS}(A[k+1 \dots n], \ell)$        ▷ search in array $A[k+1 \dots n]$, sorted in ascending order
**Output**: $t_1$ or $t_2$
```

The total runtime of the algorithm is $O(\log n + \log n + \log n)$, i.e. $O(\log n)$.

Guidelines for correction:

Award 1/2 points for each part of the exercise that is solved correctly. Do not award points for algorithms that do not meet the runtime requirements.

Exercise 4.5 *Counting function calls in loops (cont'd)* **(1 point)**.

For each of the following code snippets, compute the number of calls to f as a function of $n \in \mathbb{N}$. We denote this number by $T(n)$, i.e. $T(n)$ is the number of calls the algorithm makes to f depending on the input n. Then T is a function from $\mathbb{N}$ to $\mathbb{R}^+$. For part (a), provide **both** the exact number of calls and a maximally simplified asymptotic bound in Θ notation. For part (b), it is enough to give a maximally simplified asymptotic bound in Θ notation. For the asymptotic bounds, you may assume that $n \geq 10$.

(a)

Algorithm 4

```
$i \leftarrow 1$
**while** $i \leq n$ **do**
    $j \leftarrow 1$
    **while** $\sqrt[i]{j} \leq n$ **do**
        $f()$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
```

Hint: *You may use the formula for a finite geometric series without proof*

$$\sum_{i=0}^{n} ar^i = \frac{a(r^{n+1} - 1)}{r - 1} \text{ for } r \neq 1.$$

Solution:

The inner loop increases as long as $\sqrt[i]{j} \leq n$ which means $j \leq n^i$ and thus performs $\sum_{j=1}^{n^i} 1 = n^i$ calls to f. So in total we perform $\sum_{i=1}^{n} n^i = \sum_{i=0}^{n-1} n(n)^i = \frac{n(n^n-1)}{n-1}$ calls to f. So as

$$\begin{aligned}
\lim_{n\to\infty} \frac{\frac{n(n^n-1)}{n-1}}{n^n} &= \lim_{n\to\infty} \frac{n}{n-1} \cdot \frac{n^n-1}{n^n} \\
&= \lim_{n\to\infty} \left(1 + \frac{1}{n-1}\right) \cdot \frac{n^n-1}{n^n} = \lim_{n\to\infty} \frac{n^n-1}{n^n} + \lim_{n\to\infty} \frac{1}{n-1} \frac{n^n-1}{n^n} \\
&= \lim_{n\to\infty} \frac{n^n-1}{n^n} + 0 \\
&= \lim_{n\to\infty} 1 - \frac{1}{n^n} = 1,
\end{aligned}$$

we get this equal to $\Theta(n^n)$.

Alternatively, from the sum formula, we have

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} n^i}{n^n} = \sum_{i=1}^{n} \lim_{n\to\infty} \frac{n^i}{n^n} = \sum_{i=1}^{n} \lim_{n\to\infty} \frac{1}{n^{n-i}} = 0 + \cdots + 0 + 1 = 1$$

since all terms go to zero except $i = n$ and thus we can also conclude that this is equal to $\Theta(n^n)$.

Algorithm 5

(b)

```
function $A$(n)
    $i \leftarrow 1$
    while $i \leq n$ do
        $j \leftarrow i$
        while $j \leq n$ do
            $f()$
            $f()$
            $j \leftarrow j + 1$
        $i \leftarrow i + 1$
    $k \leftarrow \lfloor \frac{n}{2} \rfloor$
    for $\ell = 1 \ldots 3$ do
        if $k > 0$ then
            $A(k)$
```

You may assume that the function $T : \mathbb{N} \to \mathbb{R}^+$ denoting the number of calls of the algorithm to f is increasing.

Hint: *Recall exercise 0.1. If $T(n) = aT(n/2) + g(n)$ for some function $g(n)$, then find a bound $Cn^b \leq g(n) \leq C'n^b$ for two constants C and C' and then use the Θ version of the master theorem. Equivalently show that $g(n) = \Theta(n^b)$.*

Solution:

Given i, the innermost loop performs $\sum_{j=i}^{n} 2 = 2(n - i + 1)$ calls to f. Hence, the second loop (guarded by $i \leq n$) performs $\sum_{i=1}^{n} 2(n - i + 1) = 2n^2 - n(n+1) + 2n = n^2 + n$ calls to f using $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ from exercise 0.1. If $\lfloor \frac{n}{2} \rfloor = 0$ (i.e. $n = 1$), then $k = 0$, so the algorithm makes $1^2 + 1 = 2$ calls to f. Thus $T(1) = 2$. For $n \geq 2$, we have $k = \lfloor \frac{n}{2} \rfloor > 0$ and thus we get the following relation $T(n) = n^2 + n + 3T(\lfloor \frac{n}{2} \rfloor)$. For even n, this relation is $T(n) = n^2 + n + 3T(\frac{n}{2})$.

As $n^2+n \leq 2n^2$ and $n^2 \leq n^2+n$, we have $n^2 \leq n^2+n \leq 2n^2$. So

$$n^2 + 3T(\frac{n}{2}) \leq T(n) \leq 2n^2 + 3T(\frac{n}{2}).$$

So we can apply the master theorem to get a theta bound with $a = 3$, $b = 2$, $C = 2$ and $C' = 1$. We get ($2 > \log_2(3) \approx 1.584$) $T(n) = \Theta(n^2)$ for any integer $n \geq 1$ since T is increasing.

In a previous version of the sheet, we could only conclude a theta bound from the master theorem if T satisfies $T(n) = aT(n/2) + Cn^b$. The current version lets us conclude the same theta bound but under the weaker assumption on T that $aT(n/2) + C'n^b \leq T(n) \leq aT(n/2) + Cn^b$ for potentially different constants C and C' (note that the previous version is equivalent to this version with $C = C'$).

To prove the theta bound using the previous version, we can conclude an upper bound first by using the master theorem for $T(n) \leq 2n^2 + 3T(\frac{n}{2})$ to get the upper bound $T(n) \leq O(n^2)$ which holds for all $n \in \mathbb{N}$ (T is increasing).

For the lower bound, assume $n = 2^k$ for some integer $k \in \mathbb{N}$. Define $T'(n) = n^2 + 3T'(\frac{n}{2})$ with initial condition $T(1) = T'(1)$. Using the master theorem theta bound for $T'(n)$ with $a = 3$, $b = 2$ and $C = 1$ we get $T'(n) = \Theta(n^2)$ and thus by definition also $T'(n) \geq \Omega(n^2)$. Notice that $T(n) = n^2 + n + 3T(\frac{n}{2}) \geq T'(n)$, therefore $T(n) \geq T'(n) \geq \Omega(n^2)$.

To get this to hold for all $n \in \mathbb{N}$, let C be such that $T(n) \geq Cn^2$ for $n = 2^k$ where $k \in \mathbb{N}$. For n not of this form, suppose $2^k < n < 2^{k+1}$ for some $k \in \mathbb{N}$, then $2^{2k} > \frac{n^2}{4}$. So since T is increasing we have

$$T(n) \geq T(2^k) \geq C(2^k)^2 = C(2^{2k}) > C\frac{n^2}{4}$$

Thus we can conclude that $T(n) \geq \Omega(n^2)$ for all $n \in \mathbb{N}$. Combining the two bounds $T(n) \leq O(n^2)$ and $T(n) \geq \Omega(n^2)$ we can again conclude that $T(n) = \Theta(n^2)$.

(c)* Prove that the function $T : \mathbb{N} \to \mathbb{R}^+$ from the code snippet in part (b) is indeed increasing.

Hint: *You can show the following statement by mathematical induction: "For all $n' \in \mathbb{N}$ with $n' \leq n$ we have $T(n'+1) \geq T(n')$".*

Solution:

We show the statement suggested in the hint by mathematical induction.

- **Base Case.**
  We have $T(2) = 2^2 + 2 + 3T(1) = 12 \geq 2 = T(1)$, so the base case holds as the only $n' \in \mathbb{N}$ that is at most 1 is $n' = 1$.
- **Induction Hypothesis.**
  Assume that for some $k \in \mathbb{N}$ we have $T(k'+1) \geq T(k')$ for all $k' \in \mathbb{N}$ with $k' \leq k$.
- **Inductive Step.**
  We must show that $T(k+2) \geq T(k+1)$. Together with the induction hypothesis this shows that $T(k'+1) \geq T(k')$ for all $k' \in \mathbb{N}$ with $k' \leq k+1$.
  We have that
  $$\left\lfloor \frac{k+1}{2} \right\rfloor \leq k.$$
  By the induction hypothesis
  $$T\left(\left\lfloor \frac{k+2}{2} \right\rfloor\right) \geq T\left(\left\lfloor \frac{k+1}{2} \right\rfloor\right).$$

This is true since either $\lfloor\frac{k+2}{2}\rfloor = \lfloor\frac{k+1}{2}\rfloor$ or $\lfloor\frac{k+2}{2}\rfloor = \lfloor\frac{k+1}{2}\rfloor+1$. For the first case the inequality is actually an equality and the second case is covered by the induction hypothesis. Using the relation from above we get

$$T(k+2) = (k+2)^2+(k+2)+3T\left(\left\lfloor\frac{k+2}{2}\right\rfloor\right) \geq (k+1)^2+(k+1)+3T\left(\left\lfloor\frac{k+1}{2}\right\rfloor\right) = T(k+1).$$

By the principle of mathematical induction, for every $n \in \mathbb{N}$ we have for $n' \in \mathbb{N}$ with $n' \leq n$ that $T(n'+1) \geq T(n')$. In particular, $T(n+1) \geq T(n)$ is true for any $n \in \mathbb{N}$ and the function T is increasing.

Guidelines for correction:

This exercise consists of two parts (excluding part c). Award 1/2 point for each correct exercise.